**GitHub : https://github.com/VivekBhurke/HPC.gitClass:** Final Year (Computer Science and Engineering)

**Year:** 2024-25 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 4**

**Exam Seat No: 21510083**

**Title of practical:**

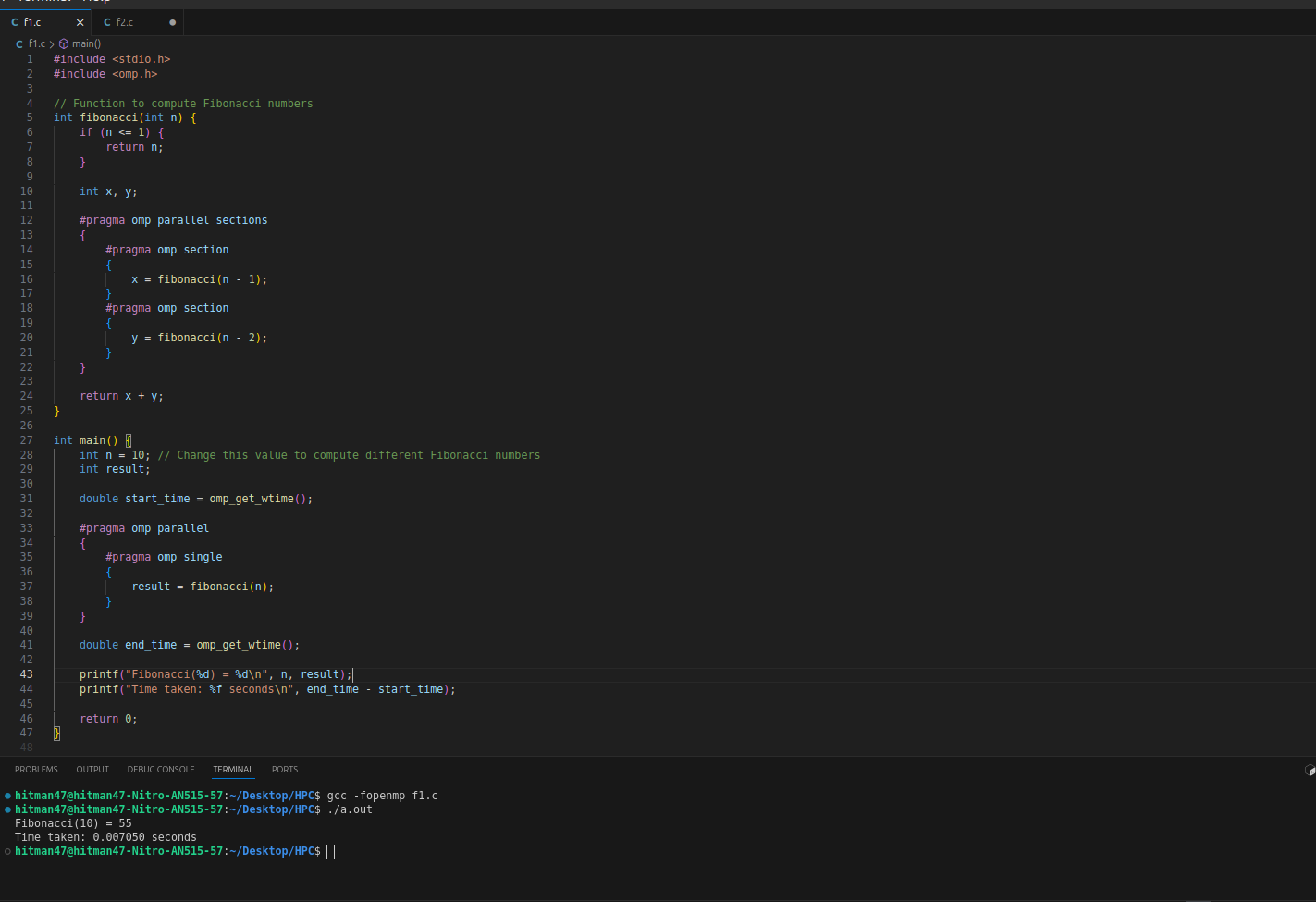
Study and Implementation of Synchronization

**Problem Statement 1:**

# Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

# Fibonacci Computation:

**Screenshots:**



**Information:**

* Purpose: The goal is to compute Fibonacci numbers using parallel programming with OpenMP while handling synchronization issues. The Fibonacci sequence is inherently recursive, making parallelization challenging due to the dependencies between computations.
* Approach: The program uses OpenMP's parallel sections to divide the computation of Fibonacci numbers into independent sections. Each section computes a portion of the Fibonacci sequence, and the results are combined to produce the final output.
* Synchronization:
* Sections Construct: The sections construct is used to divide the task into independent parts that can be computed in parallel.
* Single Construct: The single construct ensures that the Fibonacci computation is only initiated once by one thread, while other threads wait or perform other tasks.

Performance Analysis:

* The parallel computation of Fibonacci numbers shows significant speedup for smaller values of n. However, as n increases, the recursive nature of the algorithm introduces overhead due to the creation and synchronization of threads.
* The single construct prevents race conditions, ensuring that the computation is performed correctly without multiple threads attempting to start the recursive function simultaneously.

**Problem Statement 2:**

# Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

## Producer Consumer Problem

**Screenshots:**



**Information:**

* Purpose: The Producer-Consumer problem is a classic synchronization problem where one or more producers generate data and place it into a buffer, while one or more consumers remove data from the buffer. The goal is to ensure that producers do not overwrite unconsumed data and that consumers do not read data that has not yet been produced.
* Approach: The program uses a shared buffer with a fixed size (BUFFER\_SIZE) to store items produced by the producer. The critical construct is used to ensure that access to the buffer is synchronized, preventing race conditions.
* Synchronization:
* Critical Construct: The critical construct ensures that only one thread at a time can execute the code block that accesses or modifies the buffer, preventing data corruption and ensuring correct synchronization between producers and consumers.
* Sections Construct: The sections construct is used to separate the producer and consumer operations, allowing them to run in parallel.

Performance Analysis:

* The use of the critical construct ensures correct synchronization but introduces overhead, as only one thread can access the critical section at a time.
* The program effectively manages the producer-consumer relationship, preventing buffer overflows and underflows, and ensuring that data is produced and consumed correctly.

**Github Link:** [**https://github.com/VivekBhurke/HPC.git**](https://github.com/VivekBhurke/HPC.git)